Scalable Implicit Solver Interoperability via the ESI Standards Effort

Robert L. Clay
http://www.eterascale.com

18 October 2001

LACSI Symposium

Sante Fe, NM

Overview of this Presentation

- Background on the ESI effort – who, what, why
- Core object/component abstractions
  - ESI base classes (base interfaces)
  - Plug-&-play solvers and solver library interoperability
- Status and plans for the ESI effort

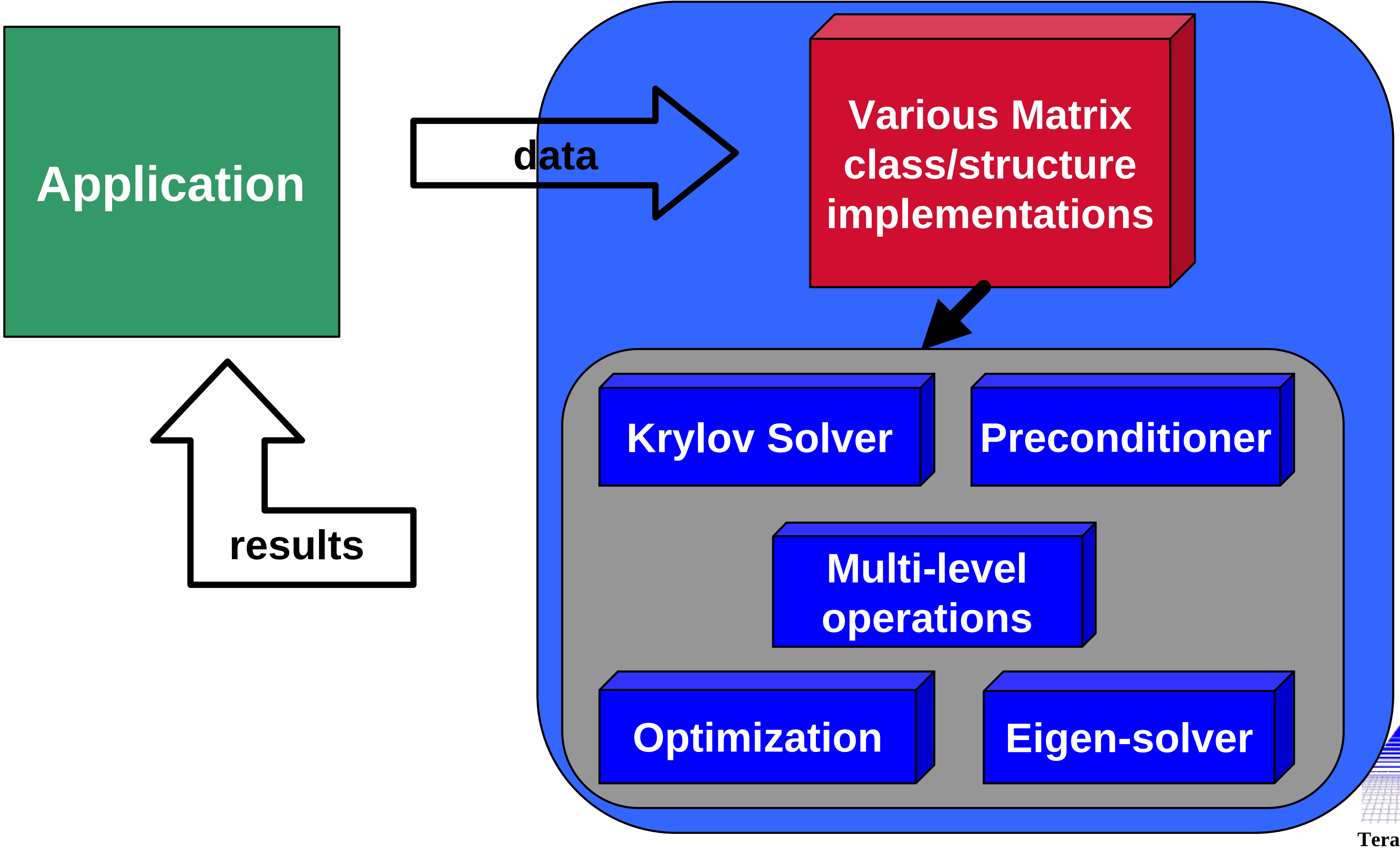
ESI Participants and URL's

- DOE, academic, and industrial initiative, building on previous efforts.
  - » ANL, LANL, LBL/NERSC, LLNL, ORNL, SNL
  - » U.C. Davis, U. Indiana, U. Utah
  - » TeraScale, LLC
- Home web site: http://z.ca.sandia.gov/esi
  - » general background and voting info
  - » email archives
- Distribution site: http://www.eterascale.com/esi
  - » CVS repository, headers, and specification
  - » Reference implementation
- Technical forum: if-forum@z.ca.sandia.gov

What’s the ESI about?

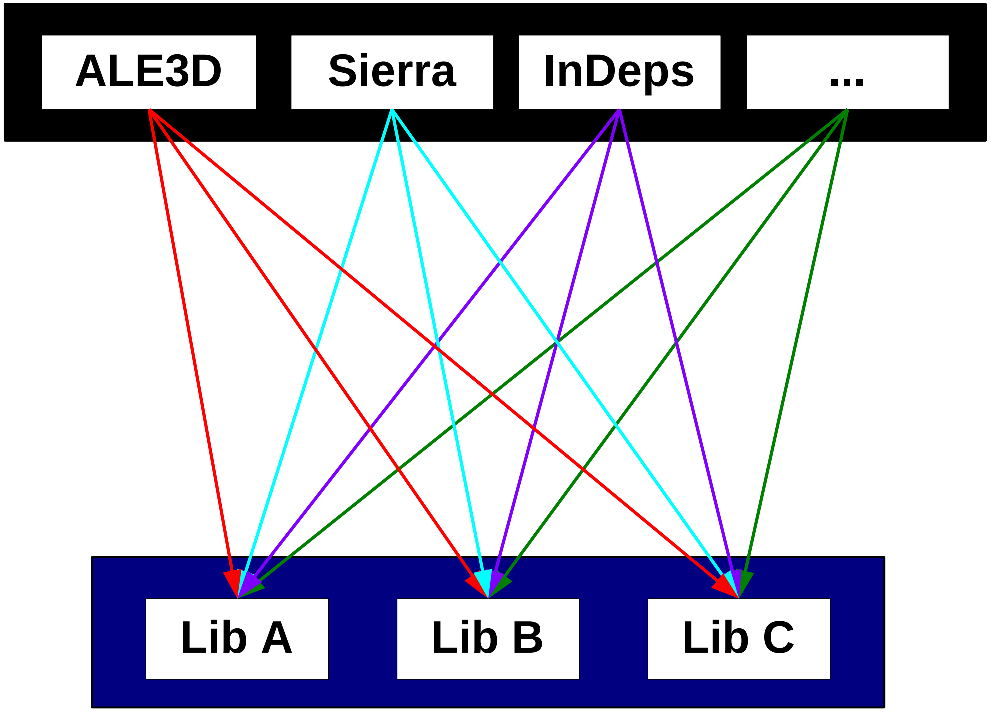
- Standards effort for interface design, specification, and prototyping:
  - » *scalable sparse linear* solution services and operators
  - » discretization abstractions (e.g., FEI, structured mesh, …)
  - » languages to include C, C++, Fortran (JAVA?)
- We are developing an integrated system of object-oriented interface specifications for shared, scalable, solver components.
- Our focus is on obtaining a long-term solution suitable for tera-scale applications (e.g., ASCI).
- We encourage participation in technical discussion and development.

Sparse Linear Algebra - do we need components?



TeraScale

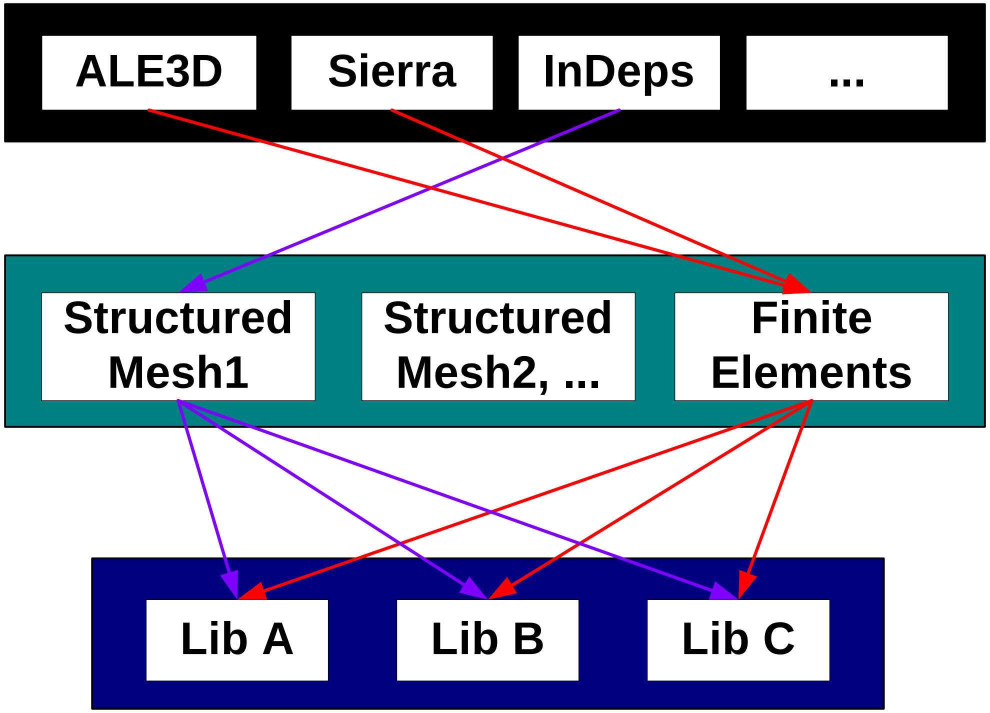
Many-Many



This is bad for users - too difficult to swap solvers,and the solvers don't interoperate.

TeraScale

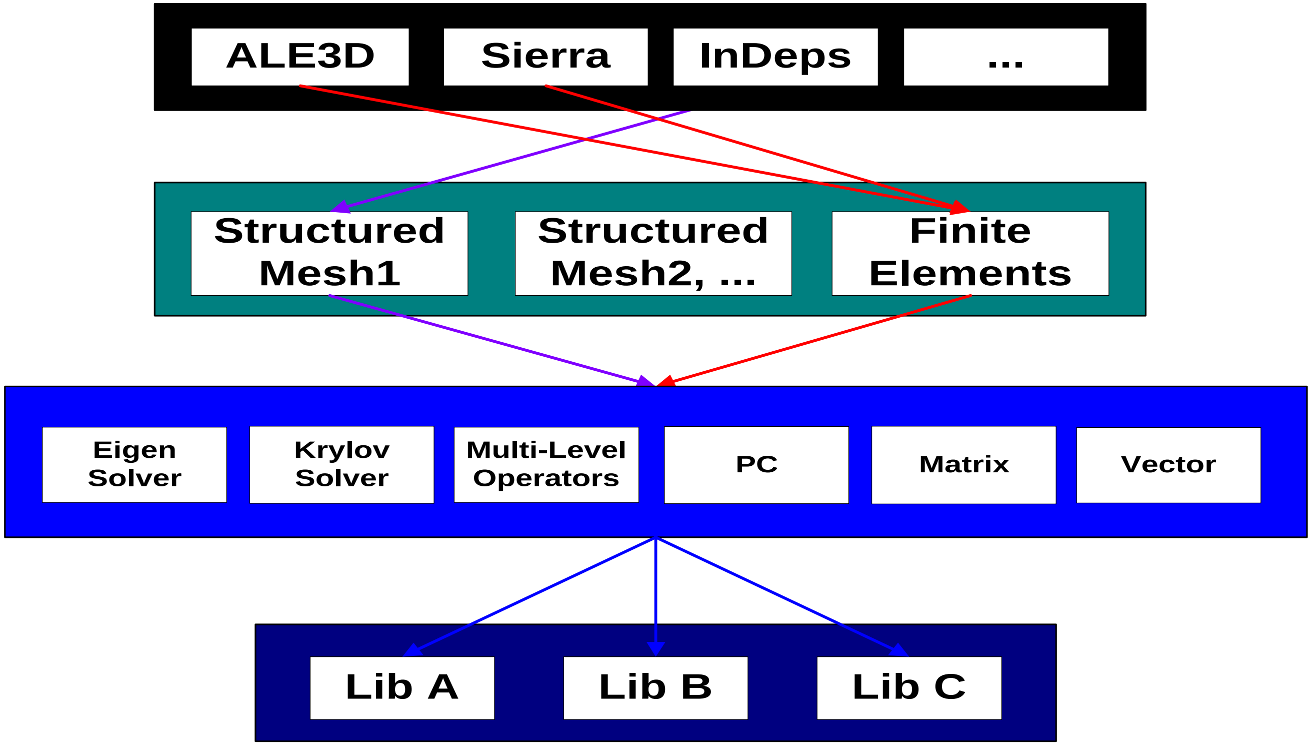
Many-1-Many



Better for users - discretization abstraction supports multiple solvers. However, solvers still don't interoperate.

TeraScale

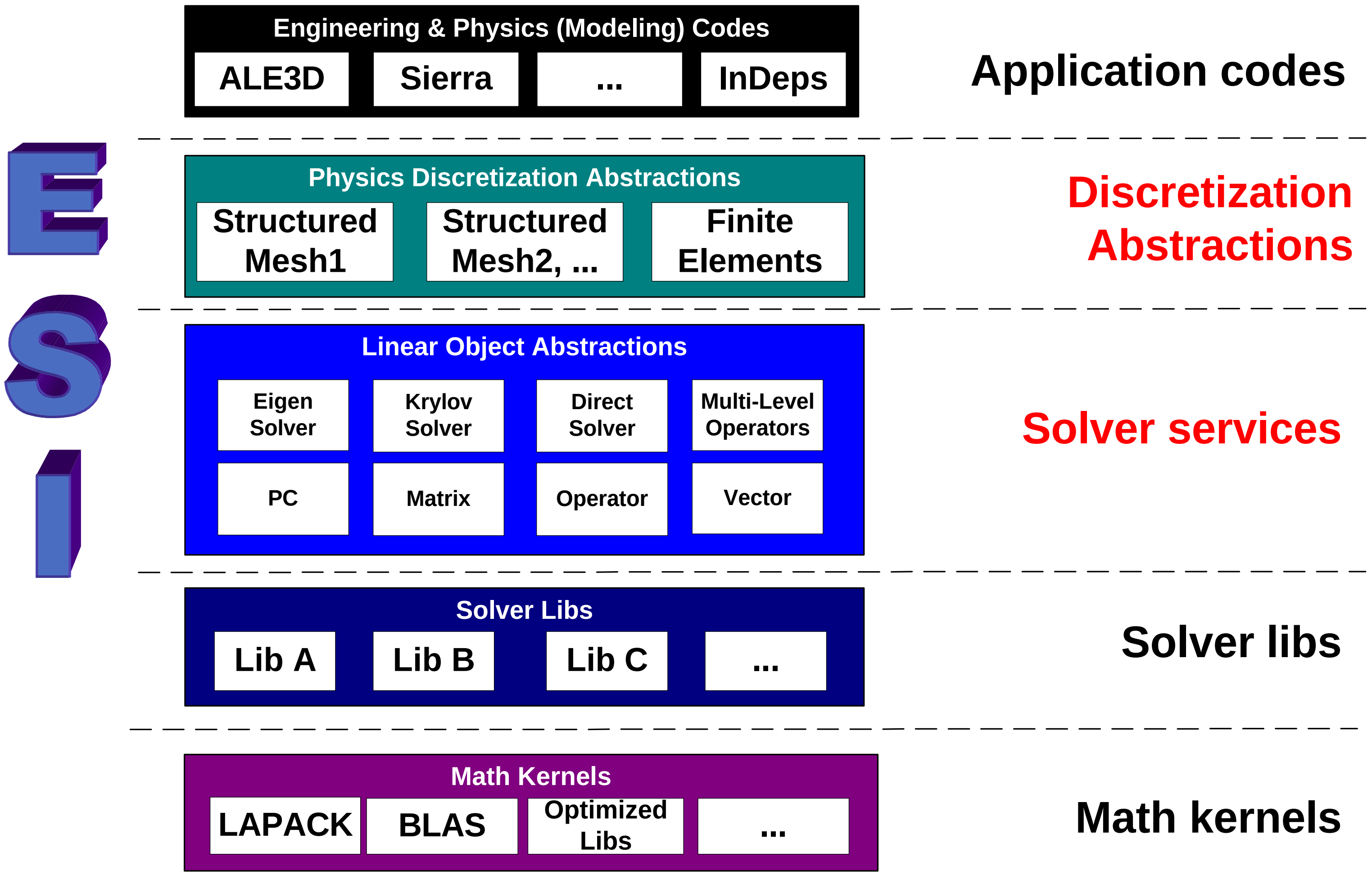
Many-1-1-Many



Better yet - discretization abstractions map into a ‘standard’ solver space, where the solvers are designed to interoperate.

TeraScale

Abstraction Layers



Engineering & Physics (Modeling) Codes

ALE3D | Sierra | ... | InDeps

Application codes

Physics Discretization Abstractions

Structured Mesh1 | Structured Mesh2, ... | Finite Elements

Discretization Abstractions

Linear Object Abstractions

Eigen Solver | Krylov Solver | Direct Solver | Multi-Level Operators

PC | Matrix | Operator | Vector

Solver services

Solver Libs

Lib A | Lib B | Lib C | ...

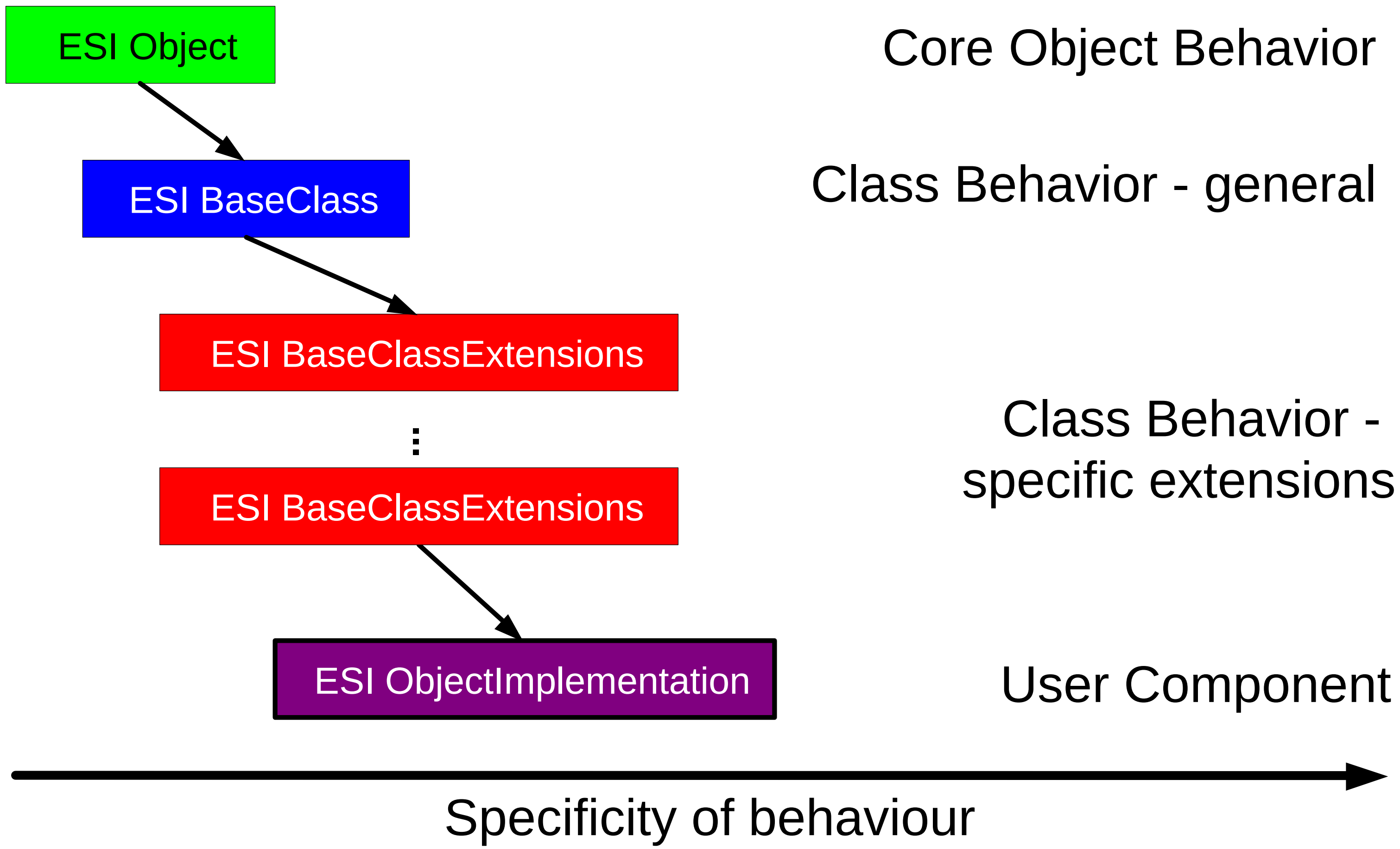
Solver libs

Math Kernels

LAPACK | BLAS | Optimized Libs | ...

Math kernels

ESI Object Model



ESI Object

ESI BaseClass

ESI BaseClassExtensions

⋮

ESI BaseClassExtensions

ESI ObjectImplementation

Core Object Behavior

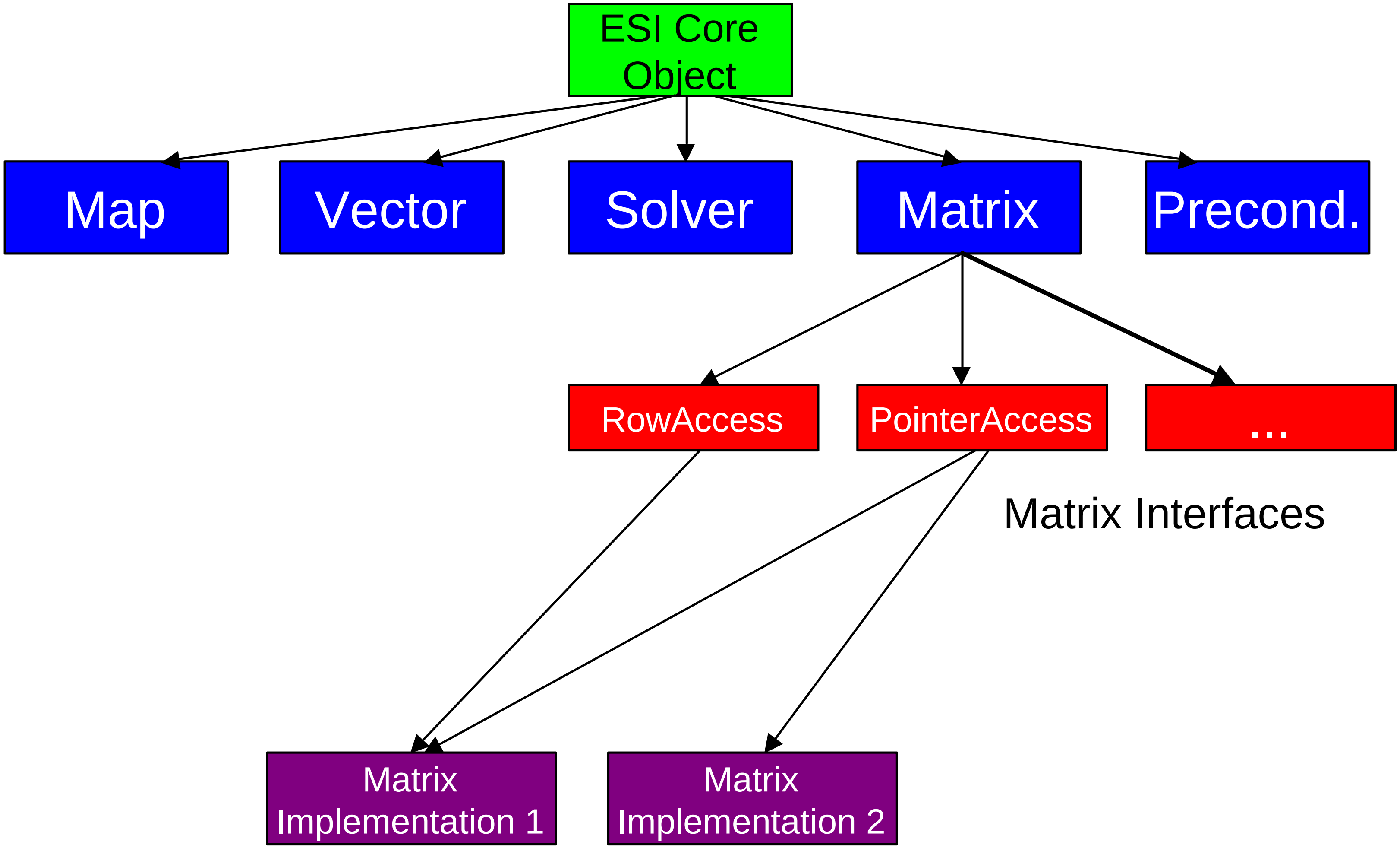
Class Behavior - general

Class Behavior - specific extensions

User Component

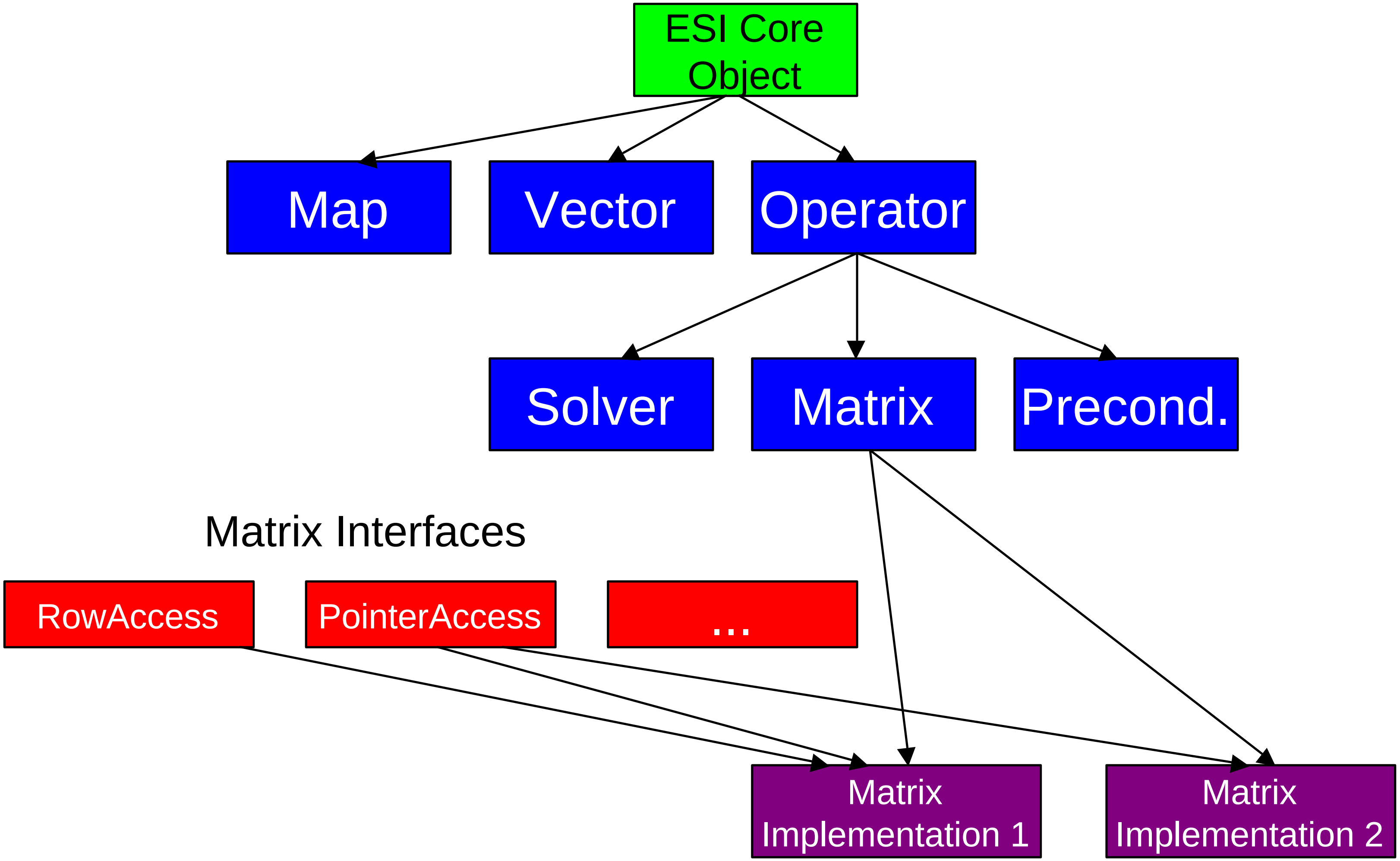
Specificity of behaviour

ESI Inheritance Scheme – Initial Proposal -



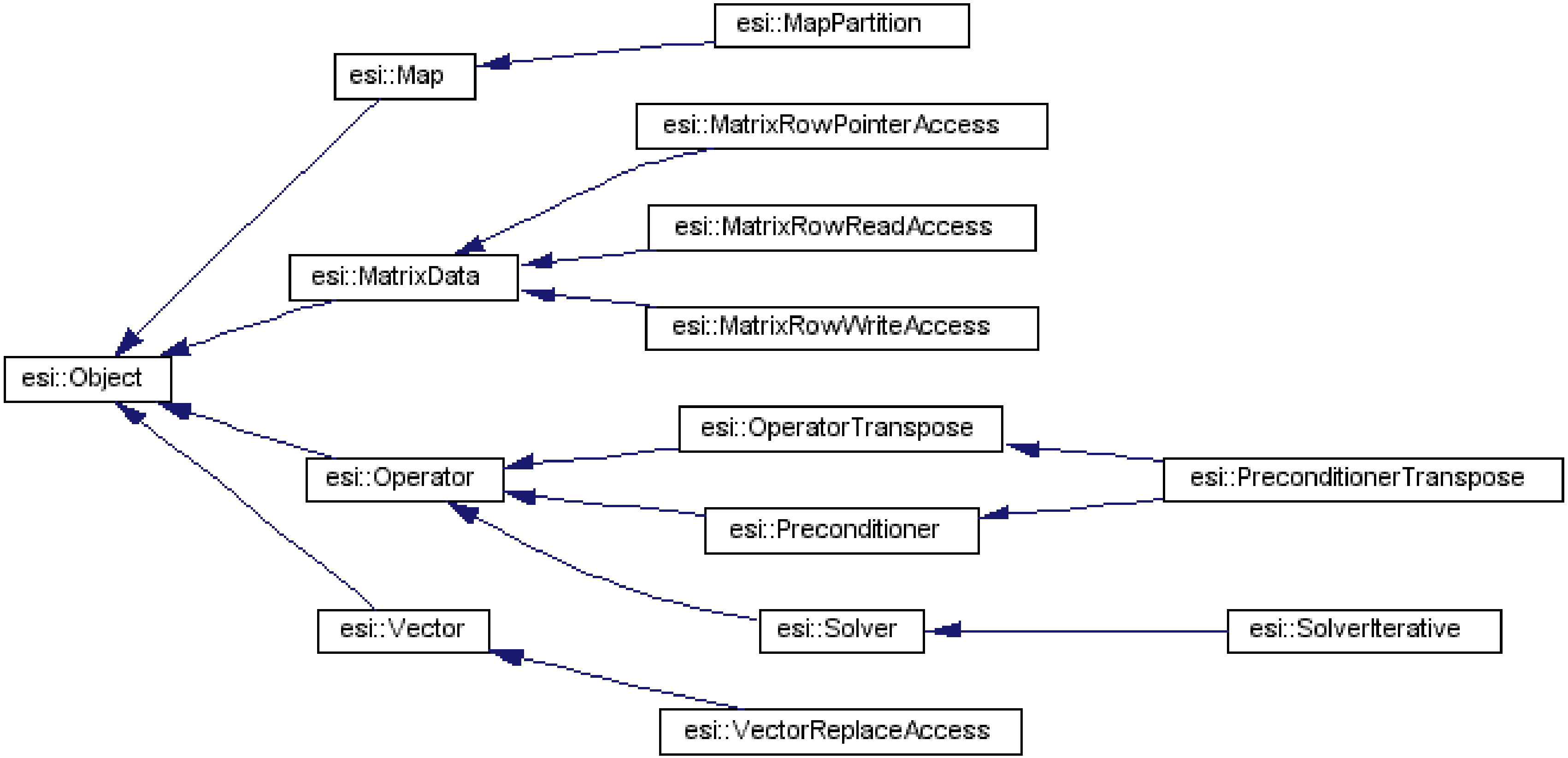
TeraScale

ESI Inheritance Scheme - Next Proposal -



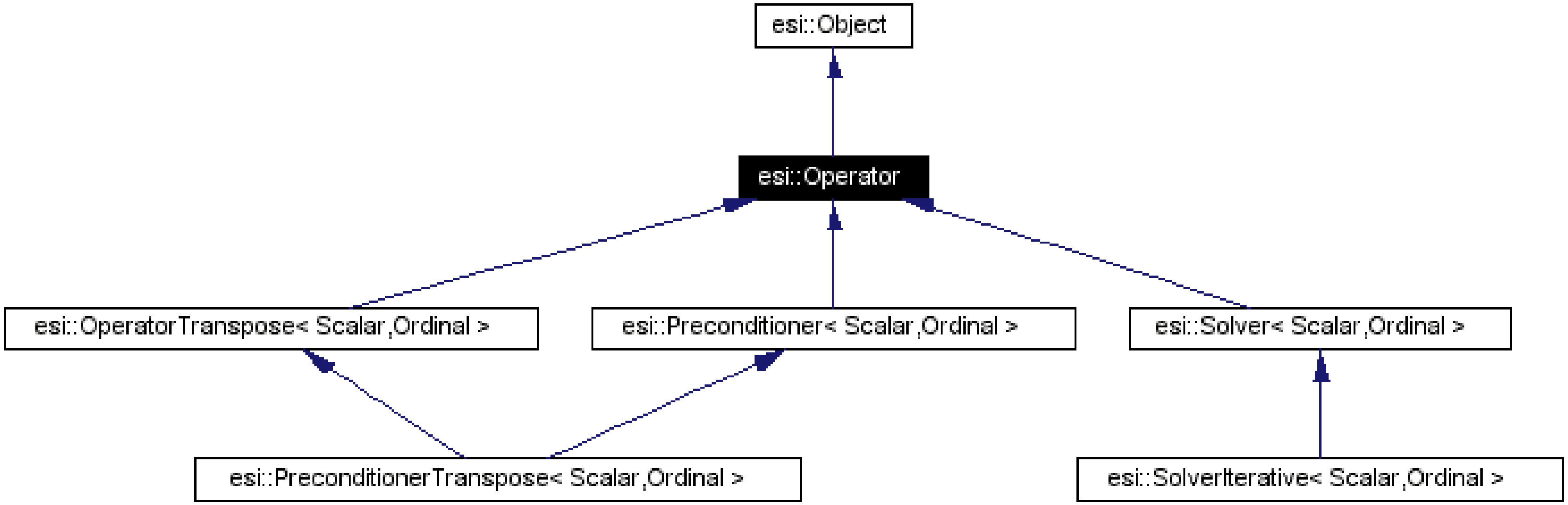
TeraScale

ESI Inheritance Scheme - Actual



TeraScale

Multiple Inheritance – good, bad, or irrelevant?

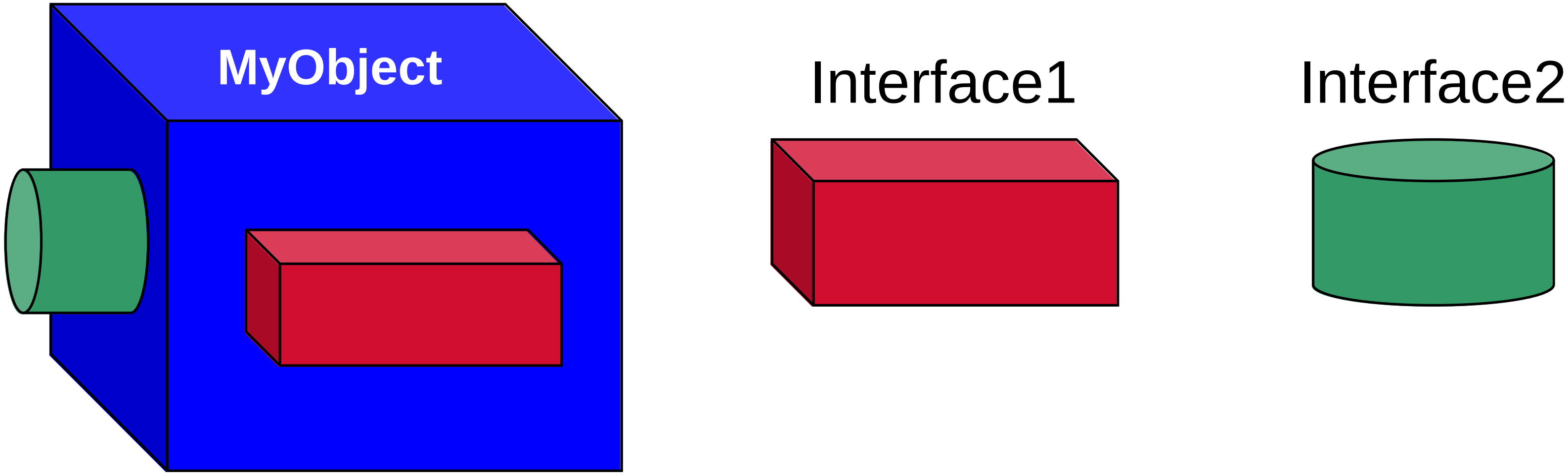

esi::Object
esi::Operator
esi::OperatorTranspose< Scalar,Ordinal >
esi::Preconditioner< Scalar,Ordinal >
esi::Solver< Scalar,Ordinal >
esi::PreconditionerTranspose< Scalar,Ordinal >
esi::SolverIterative< Scalar,Ordinal >

- Why not have OperatorTranspose peer to Operator?
  - OperatorTranspose implementations will have both 'apply' and 'applyTranspose'.
  - Similar situation for Preconditioner classes.

TeraScale

Multiple Inheritance -- think of building-blocks

Inheriting multiple standard interfaces makes **MyObject** “plug-compatible” in multiple roles.

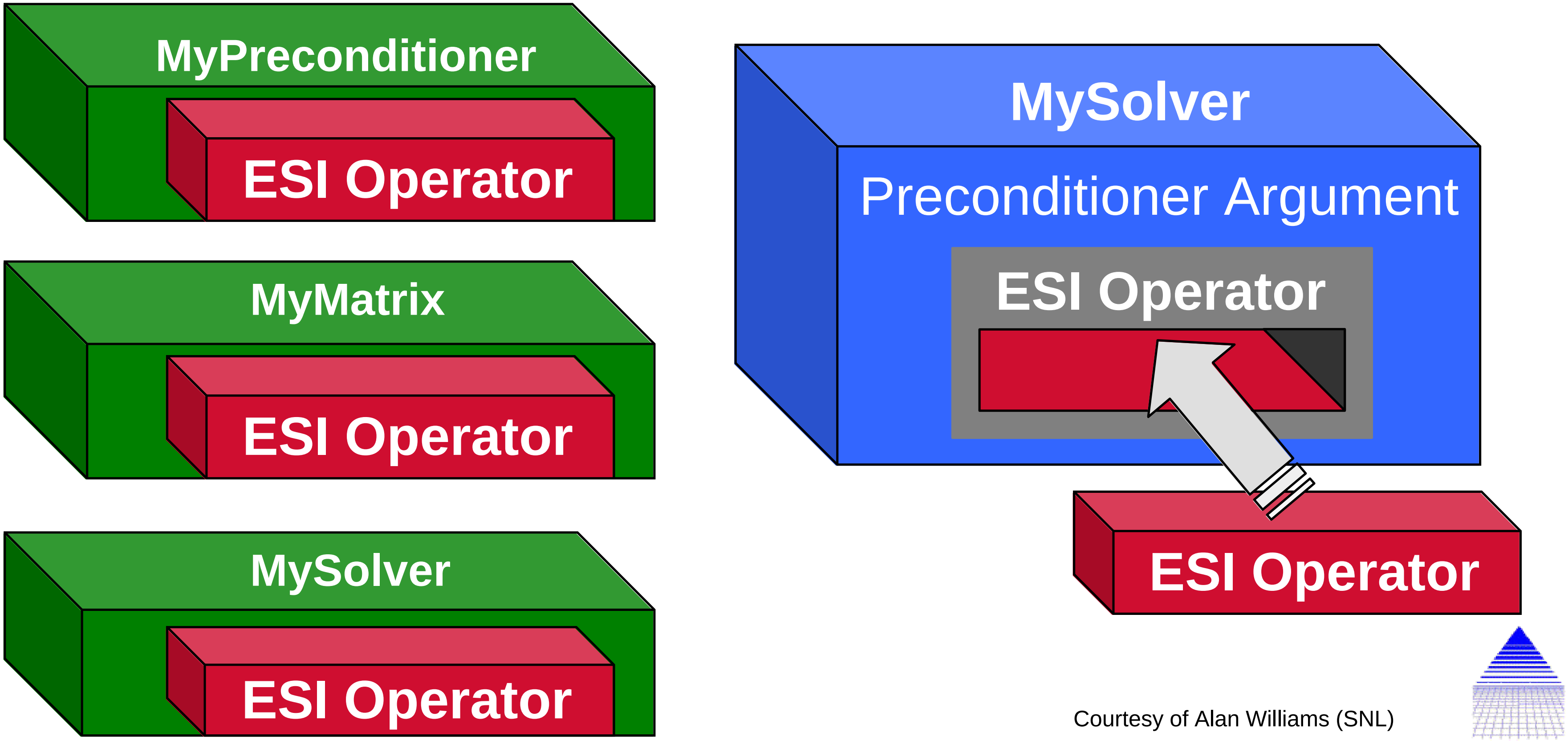


Courtesy of Alan Williams (SNL)

TeraScale

Preconditioning a Solver with an ESI Operator

Anything that supplies ESI Operator functionality can be plugged in and used as a preconditioner.



Courtesy of Alan Williams (SNL)

TeraScale

ESI API characteristics

- Strengths
  - ‘standard’, library-neutral interfaces – ESI compliant implementations are interoperable, even if they come from different libraries
  - flexible, modular design – lots of possible combinations
  - binding to existing libs is straightforward
  - ‘component’ ready – abstract interfaces w/ ‘query interface’ method
  - templated types (‘complex’ type support ‘easy’)
  - extensible
  - solver developers can extend their list of users via ESI bindings
  - solver users can extend their list of solvers available through one API
- Weaknesses
  - C++ only today – SIDL mapping close behind (C, Fortran, Java?)
  - initial complexity of using the interfaces rather high (due to templated types and multiple inheritance) – reference implementations help
  - ‘common’ model/API can’t capture (and isn’t intended to either) the full richness of all solver packages

Current Status

- ESI v1.0 interfaces have passed the first vote, and second vote is expected to pass w/in a month
  - It took ~3 years to get to the 1.0 spec!
  - http://www.eterascale.com/esi has current distribution, including reference implementation.
  - Standard’s group is focused on wrapping up v1.0 release and polishing up the package bindings and reference implementation.

- Work is underway or complete on several packages:
  - TSC reference implementation (RLC)
  - ISIS++ bindings (ABW/BA)
  - Trilinos bindings (ABW/MH)
  - QMRPACK bindings (NN/MG/RLC)
  - SPOOLES 2.2 bindings (RLC/CA/MG/NN)
  - PETSc (BS/SB/LCM)

Plans

- We’re working on a set of interfaces for FE / solver coupling – derivative work of the FEI 2.0
  - http://www.eterascale.com/fei-lite
  - Modularized vs. monolithic FEI 2.0
  - Augments ESI interfaces
  - Could be an FEI 3.0, but will probably be proposed as part of the ESI spec instead.
- We’re also working on block matrix/vector abstractions:
  - Hybrid block matrix/vector classes
  - Generalized/derived from Daniel White’s 2x2 hybrid block classes.
- Other abstractions we need to address:
  - Structured mesh ‘loading’
  - Eigen solver abstractions
  - Better ways of handling ‘exceptions’
  - Multi-level methods